CLASSIC-ADA™

Lois Valley
Software Productivity Solutions, Inc.

The SPS product, Classic-Ada™, is a software tool that supports object-oriented Ada programming with powerful inheritance and dynamic binding. Object Oriented Design (OOD) is an easy, natural development paradigm, but it is not supported by Ada. Following the DOD Ada mandate, SPS developed Classic-Ada to provide a tool which supports OOD and implements code in Ada. It consists of a design language, a code generator and a toolset. As a design language, Classic-Ada supports the object-oriented principles of information hiding, data abstraction, dynamic binding, and inheritance. It also supports natural reuse and incremental development through inheritance, code factoring, and Ada, Classic-Ada, dynamic binding and static binding in the same program. Only nine new constructs were added to Ada to provide object-oriented design capabilities. The Classic-Ada code generator translates user application code into fully compliant, ready-to-run, standard Ada. The Classic-Ada toolset is fully supported by SPS and consists of an object generator, a builder, a dictionary manager, and a reporter. Demonstrations of Classic-Ada and the Classic-Ada Browser were given at the workshop.

Why *Classic-Ada*™?

- Ada Mandate
- Object-Oriented Design is an easy natural development paradigm
- Ada doesn't support the object-oriented paradigm
- SPS needed a tool that allowed us to think in OOD and implement in Ada
- *Classic-Ada* is our answer to that need

What is *Classic–Ada*™?

Classic–Ada is:

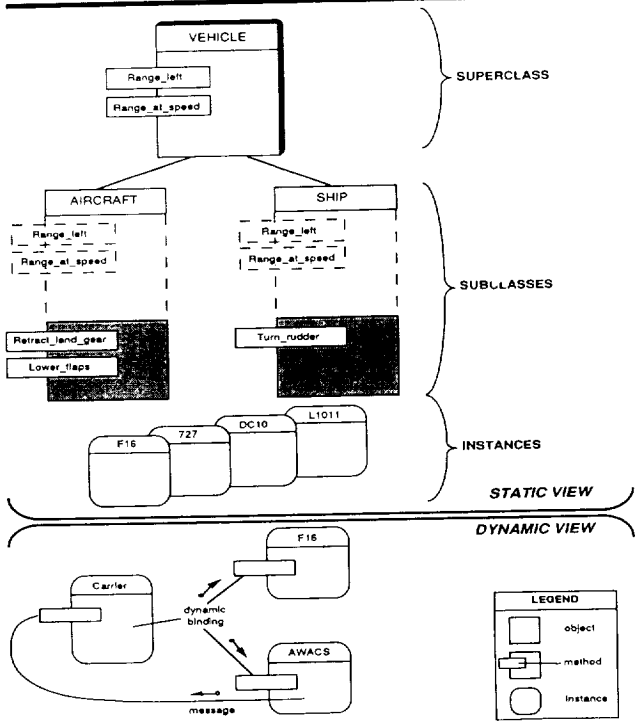
- A design language
- A code generator
- A toolset

Classic-Ada™ as a Design Language

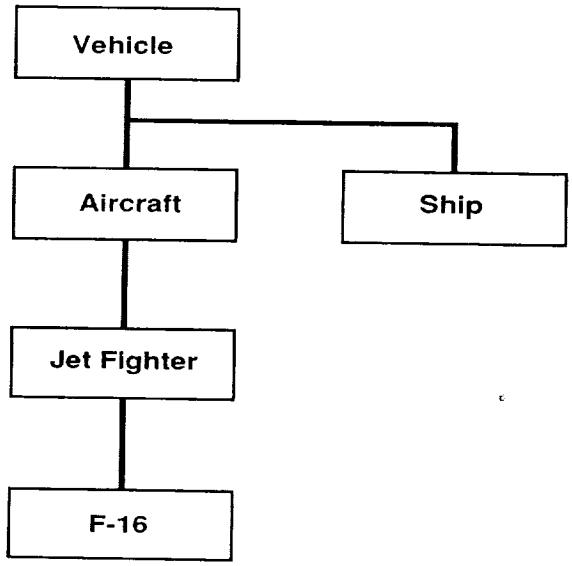
Supports object-oriented principles

- Information hiding - hiding the state of software components in variables visible only within the scope of that component
- Data abstraction - abstract data types defining an internal representation plus a set of operations used to access and manipulate it
- Dynamic binding - determining which operation is invoked for a specific abstract data type dynamically at runtime, depending on the object being manipulated
- Inheritance - enabling the easy creation of objects that are almost like other objects with just a few changes

Inheritance and Dynamic Binding



Inheritance Hierarchy



Reusability

- Inheritance enables the creation of objects that are *almost like* other objects with just a few changes
- Generalization promotes the constant migration to more general (and more reusable) objects
- Inheritance enhances *code factoring*, i.e. code to perform a particular task is found in only one place
- Dynamic binding increases flexibility by allowing the addition of new object classes without modifying or recompiling existing code
- Polymorphism, the ability for different classes to respond to the same message promotes interchangeable parts

Natural Reuse

"Object-oriented development integrates reuse into the development process so well that developers will find themselves developing reusable objects and reusing existing objects without even thinking about it."

Smaller, Cheaper Solutions

- Solve large problems by making the solutions smaller
  - Typically at least 1/4 the number of lines of code in an OOPL
  - Often as little as 1/10 or 1/20
- Productivity increases because effort per line of code is about the same as with procedural HOLs
- Manageability improves dramatically
  - Software system is easier to understand
  - There are less people to manage

  "Managers must reward designers for doing less - not more."

  -- Wilf LaLonde
- As the development converges, the lines of code will actually decrease as generalizations further optimize and compress the code
- You don't have to do *programming-in-the-large* to solve large programs if you make the large program small

Large OOP Experiences

500,000 - 1,000,000 LOC ► 25,000 - 100,000 LOC
Procedural HOL OOPL

- Operating systems
- Workstation / office automation environments
- CAD /CAE
- Telecommunications
- User interface / application frameworks
- Object-bases
- Simulations
- Manufacturing, operations and control systems
- Management information systems

Classic-Ada™ as a Design Language

Has added only nine constructs to Ada to provide this powerful capability. These constructs are:

- Class
- Superclass
- Instance
- Instantiate
- Method
- Destroy
- Send
- Self
- Super

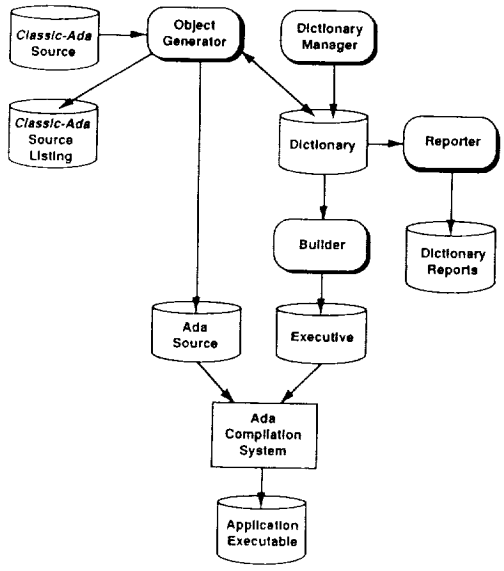
Classic-Ada™ as a Code Generator

- Generates user application code
- Generates application executive
- Generates fully compliant, ready to run, DoD standard Ada

Classic-Ada™ as a Toolset

- Is fully supported by SPS
- Consists of the following tools:
  - an object generator
  - a builder
  - a dictionary manager
  - a reporter

Classic-Ada™ Toolset



Classic-Ada™ as a Design Language

- Supports natural reuse through its inheritance capabilities
- Supports incremental development through inheritance
- Supports code factoring
- Supports Ada, *Classic-Ada*, Dynamic binding, and static binding in the same program
- Makes it easy to both generalize and specialize during development due to the way *Classic-Ada* is implemented
- Minimizes the need to compile large portions of code